第3章 程序设计初步

3.1 堆栈的作用

3.2 算术逻辑运算指令

3.3 分支程序设计

3.4 循环程序设计

3.5 子程序设计

3.2 算术逻辑运算指令

3.2.1 乘除运算指令

3.2.2 逻辑运算指令

3.2.2 移位指令

3.2.1 乘除运算指令

➢乘除运算指令

- ✓ 乘除运算指令区分有符号数与无符号数
- ✓ 对状态标志的影响，不够自然。

MUL （ unsigned MULtiply ）

IMUL （ sIgned MULtiply ）

DIV （ DIVide ）

IDIV （ sIgned DIVide ）

根据有符号与无符号，分为两组

3.2.1 乘除运算指令

➢演示函数cf38

```
int  cf38( int x, int y )
{
    return  (x * x + 3) / (168 * y);
}
```

速度最大化编译：

```
push   ebp
mov    ebp, esp
mov    eax, DWORD PTR [ebp+8]     ;取出参数x
mov    ecx, DWORD PTR [ebp+12]    ;取出参数y
imul   eax, eax                   ;实现x*x，保存在EAX
imul   ecx, 168                   ;实现168*y，保存在ECX
add    eax, 3                     ;实现x*x+3
cdq                  ;把EAX符号扩展到EDX（形成64位被除数）
idiv   ecx           ;除运算，EDX及EAX是64位被除数,ECX是除数
pop    ebp
ret
```

3.2.1 乘除运算指令

➢无符号数乘法指令（unsigned MULtiply）

✓MUL指令的一般格式

```
MUL   OPRD
```

- 指令实现两个无符号操作数的乘法运算。
- 乘数是OPRD，被乘数位于AL、AX或EAX中（由OPRD的尺寸决定，乘数和被乘数的尺寸一致）。
- 乘积尺寸翻倍：16位乘积送到AX；32位乘积送DX:AX；64位乘积送EDX:EAX。
- 操作数OPRD可以通用寄存器，可以存储单元，但不能是立即数。

3.2.1 乘除运算指令

➢无符号数乘法指令（unsigned MULtiply）

✓MUL指令的一般格式

```
MUL    OPRD
```

✓使用举例

```
MUL    BL          ; 8位乘    AL*BL    乘积 AX
MUL    DX          ;16位乘    AX*DX    乘积 DX:AX
MUL    ECX         ;32位乘    EAX*ECX  乘积 EDX:EAX
```

3.2.1 乘除运算指令

➢有符号数乘法指令（sIgned MULtiply）

✓IMUL指令的一般格式

```
IMUL   OPRD
IMUL   DEST，SRC
IMUL   DEST，SRC1，SRC2
```

有符号乘法指令
有三种形式

3.2.1 乘除运算指令

➢有符号数乘法指令（sIgned MULtiply）

✓IMUL指令的一般格式（一）

单操作数形式

IMUL　OPRD

- 单操作数乘法指令实际上有一个隐含的操作数，位于AL、AX或EAX中（取决于操作数OPRD的尺寸）。
- 它把被乘数和乘数均作为有符号数。
- 其他与无符号乘法指令MUL类似。

```
IMUL    CL
IMUL    DWORD PTR [EBP+12]        ;双字存储单元
```

3.2.1 乘除运算指令

➢有符号数乘法指令（sIgned MULtiply）

✓IMUL指令的一般格式（二）

双操作数形式

IMUL DEST，SRC

DEST <= DEST * SRC

- 目的操作数DEST只能是16位或者32位通用寄存器。
- 源操作数SRC可以是通用寄存器或存储单元（须与目的操作数尺寸一致），可以是一个立即数（尺寸不能超过目的操作数）。

3.2.1 乘除运算指令

➢有符号数乘法指令（sIgned MULtiply）

✓IMUL指令的一般格式（三）

三操作数形式

IMUL DEST，SRC1，SRC2

DEST <= SRC1 * SRC2

- 目的操作数DEST只能是16位或32位通用寄存器。
- 源操作数SRC1可以是通用寄存器或存储单元（须与目的操作数尺寸一致），但不能是立即数。源操作数SRC2只能是一个立即数（尺寸不能超过目的操作数）。

3.2.1 乘除运算指令

➢有符号数乘法指令（sIgned MULtiply）

✓使用举例

```
IMUL    BX
IMUL    EBX, ECX
IMUL    AX, CX, 3
IMUL    EDX, DWORD PTR [ESI], 5
IMUL    AX, 7
IMUL    AX, AX, 7
```

IMUL OPRD

IMUL DEST，SRC

IMUL DEST，SRC1，SRC2

3.2.1 乘除运算指令

➢无符号数除法指令（DIVide）

✓DIV指令的一般格式

```
DIV   OPRD
```

- 指令实现两个无符号操作数的除法运算。
- 除数是OPRD。被除数位于AX、DX:AX或EDX:EAX中（由OPRD的尺寸决定，被除数的尺寸翻倍）。
- 商在AL、AX或者EAX中；余数在AH、DX或者EDX中（商和余数的尺寸与oprd相同）。
- 操作数OPRD可以是通用寄存器，可以是存储单元，但不能是立即数。

3.2.1 乘除运算指令

➢无符号数除法指令（DIVide）

✓DIV指令的一般格式

```
DIV   OPRD
```

✓使用举例

```
DIV    BL                ;除数8位
DIV    CX                ;除数16位
DIV    ESI               ;除数32位
```

3.2.1 乘除运算指令

➢无符号数除法指令（DIVide）

✓DIV指令的一般格式

```
DIV    OPRD
```

✓注意：必须防止除溢出！

```
MOV    AX, 600
MOV    BL, 2
DIV    BL
```

3.2.1 乘除运算指令

➢有符号数除法指令（sIgned DIVide）

✓IDIV指令的一般格式

IDIV　OPRD

必须防止溢出！

- 指令实现两个有符号操作数的除法运算。
- 除数是OPRD。被除数位于AX、DX:AX或EDX:EAX中。商在AL、AX或者EAX中；余数在AH、DX或者EDX中。尺寸由除数OPRD决定。
- 操作数OPRD可以是通用寄存器，可以是存储单元，但不能是立即数。
- 如果不能整除，余数的符号与被除数一致，而且余数的绝对值小于除数的绝对值。

3.2.1 乘除运算指令

➢符号扩展指令

✓字节转换为字指令CBW(Convert Byte to Word)

CBW

指令把AL中的符号扩展到AH。

若AL的最高有效位为0，则AH=0；若AL的最高有效位为1，则AH=0FFH，也即AH的8位全都为1。

✓使用举例

```
MOV    AX, 3487H          ;AX=3487H
CBW                       ;AX=FF87H
MOV    AX, 8734H
CBW                       ;AX=0034H
```

3.2.1 乘除运算指令

➢符号扩展指令

✓字转换为双字指令CWD(Convert Word to Double word)

CWD

指令把AX中的符号扩展到DX。

若AX的最高有效位为0，则DX=0；若AX最高有效位为1，则DX=0FFFFH，也即DX的16位全都为1。

✓使用举例

```
MOV    AX, 3487H          ;AX=3487H
CWD                       ;DX=0000H, AX=3487H
MOV    AX, 8734H
CWD                       ;DX=FFFFH, AX=8734H
```

3.2.1 乘除运算指令

➢符号扩展指令

✓双字转换为四字指令CDQ(Convert Doubleword to quadword)

CDQ

指令把EAX中的符号扩展到EDX。

EAX的最高有效位为0，则EDX=0；若EAX最高有效位为1，则EDX=0FFFFFFFFH，也即EDX的32位全都为1。

✓使用举例

```
MOV    EAX, 12563487H    ;EAX=12563487H
CDQ                      ;EDX=00000000H, EAX=12563487H
```

3.2.1 乘除运算指令

➢符号扩展指令

✓另一条字转换为双字指令CWDE(Convert Word to Doubleword)

CWDE

指令把AX中的符号扩展到EAX的高16位。

AX的最高有效位为0，则EAX的高16位都为0；若AX的最高有效位为1，则EAX的高16位都为1。

✓使用举例

```
MOV    AX, 3487H          ;AX=3487H
CWDE                      ;EAX=00003487H
MOV    AX, 8734H
CWDE                      ;EAX=FFFF8734H
```

3.2.1 乘除运算指令

➢演示程序dp39

演示除法指令和符号扩展指令的使用

```
#include
int  main
{   int
    _asm
        MOV   AX ,-601
        MOV   BL,10
        IDIV  BL                    //除数是BL，被除数是AX
        MOV   BL,AH                 //先临时保存余数
        ;
        CBW                         //商在AL，符号扩展到AX
        CWDE                        //AX符号扩展到EAX
        MOV   quotient, EAX
        ;
        MOV   AL,BL                 //余数送到AL
        CBW                         //AL符号扩展到AX
        CWDE                        //AX符号扩展到EAX
        MOV   remainder, EAX
    }
    printf("quotient= %d\n", quotient);     //显示为-60
    printf("remainder= %d\n", remainder);   //显示为-1
    printf("\n");
```

3.2.1 乘除运算指令

➢演示程序dp39

```
_asm {
    MOV   AX ,-601
    CWD                          //AX符号扩展到DX
    MOV   BX,3
    IDIV  BX                     //除数是BX，被除数是DX:AX
    ;
    CWDE                         //商在AX，符号扩展到EAX
    MOV   quotient,EAX
    ;
    MOV   AX,DX                  //余数DX送到AX
    CWDE                         //符号扩展到EAX
    MOV   remainder,EAX
}
    printf("quotient= %d\n", quotient);      //显示为-200
    printf("remainder= %d\n", remainder);    //显示为-1
    return  0;
}
```

3.2.1 乘除运算指令

➢符号扩展传送指令MOVSX（Move with Sign-Extension）

✓MOVSX指令的一般格式

```
MOVSX  DEST，SRC
```

- 指令把源操作数SRC符号扩展后送至目的操作数DEST。
- 源操作数SRC可以是通用寄存器或存储单元，而目的操作数DEST只能是通用寄存器。
- 目的操作数的尺寸必须大于源操作数的尺寸。源操作数的尺寸可以是8位或者16位；目的操作数的尺寸可以是16位或者32位。

3.2.1 乘除运算指令

➢符号扩展传送指令MOVSX（Move with Sign-Extension）

✓MOVSX指令的一般格式

```
MOVSX  DEST，SRC
```

✓使用举例

```
MOV     AL, 85H          ;AL=85H
MOVSX   EDX, AL          ;EDX=FFFFFF85H
MOVSX   CX, AL           ;CX=FF85H
MOV     AL, 75H          ;AL=75H
MOVSX   EAX, AL          ;EAX=00000075H
```

3.2.1 乘除运算指令

➢符号扩展传送指令MOVSX（Move with Sign-Extension）

✓MOVSX指令的一般格式

```
MOVSX  DEST，SRC
```

✓MOVSX与CBW、CWDE

```
MOV     AL, 85H        ;AL=85H
MOVSX   AX, AL         ;AX=FF85H
等价于
MOV     AL, 85H
CBW
```

```
MOV     AX, 8500H      ;AX=8500H
MOVSX   EAX,           ;EAX=FFFF8500H
等价于
MOV     AX, 8500H
CWDE
```

3.2.1 乘除运算指令

➢零扩展传送指令MOVZX（Move with Zero-Extend）

✓MOVZX指令的一般格式

```
MOVZX   DEST，SRC
```

- 指令把源操作数SRC零扩展后送至目的操作数DEST。
- 源操作数SRC可以是通用寄存器或存储单元，而目的操作数DEST只能是通用寄存器。源操作数的尺寸可以是8位或者16位；目的操作数的尺寸只可以是16位或者32位。

✓使用举例

```
MOV     DX, 8885H              ;DX=8885H
MOVZX   ECX, DL                ;ECX=00000085H
MOVZX   EAX, DX                ;EAX=00008885H
```

3.2.1 乘除运算指令

➢演示函数cf310

```
int  cf310(char x, char y)
{
    return  ( x + 22 ) / y ;
}
```

有符号
字符型

```
push    ebp
mov     ebp, esp
movsx   eax, BYTE PTR [ebp+8]    ;把参数x符号扩展后送到eax
add     eax, 22
movsx   ecx, BYTE PTR [ebp+12]   ;把参数y符号扩展后送到ecx
cdq                              ;符号扩展，形成64位的被除数
idiv    ecx
pop     ebp
ret
```

有符号

3.2.1 乘除运算指令

➢演示函数cf311

```
unsigned int  cf311(unsigned char x, unsigned char y)
{
    return  (unsigned)( x + 22 ) / y ;
}
```

无符号
字符型

```
push    ebp
mov     ebp, esp
movzx   eax, BYTE PTR [ebp+8]    ;把参数x零扩展后送到eax
add     eax, 22
movzx   ecx, BYTE PTR [ebp+12]   ;把参数x零扩展后送到ecx
xor     edx, edx                 ;零扩展，形成64位的被除数
div     ecx
pop     ebp
ret
```

无符号

3.2.2 逻辑运算指令

➢逻辑运算指令

✓ C语言中有一组按位逻辑运算符

- 按位取反运算符 ~
- 按位与运算符 &
- 按位或运算符 |
- 按位异或运算符 ^

✓ 处理器提供一组逻辑运算指令

- 否指令 **NOT**
- 与指令 **AND**
- 或指令 **OR**
- 异或指令 **XOR**

3.2.2 逻辑运算指令

➢演示函数cf312

演示逻辑运算符的使用

```
unsigned int  cf312(unsigned int x, unsigned int y)
{
    int  z = 0;
    if ( ( x & 3 ) || ( ( x - 5 ) | ~y ) )
    {
        z = x ^ 255;
    }
    return  z;
}
```

无符号整型

3.2.2 逻辑运算指令

➢演示函数cf312

禁止优化
编译：

```
push    ebp
mov     ebp, esp                        ;建立堆栈框架
push    ecx                             ;安排局部变量z
mov     DWORD PTR [ebp-4], 0            ; z=0;
                                        ;
mov     eax, DWORD PTR [ebp+8]          ;取出参数x
and     eax, 3                          ; x & 3
jne     SHORT  LN1cf312                 ;结果不为0，条件成立，跳转
                                        ;
mov     ecx, DWORD PTR [ebp+8]          ;又取出参数x
sub     ecx, 5                          ; x - 5
mov     edx, DWORD PTR [ebp+12]         ;取出参数y
not     edx                             ; ~y
or      ecx, edx                        ;( x - 5 ) | ~y
je      SHORT  LN2cf312                 ;结果为0，条件不成立，跳转
```

3.2.2 逻辑运算指令

➢演示函数cf312

续前页

```
LN1cf312:                                   ;条件成立
    mov    eax, DWORD PTR [ebp+8]           ;又取出参数x
    xor    eax, 255                          ; x ^ 255
    mov    DWORD PTR [ebp-4], eax           ;保存x
                                            ;
LN2cf312:                                   ;准备返回
    mov    eax, DWORD PTR [ebp-4]           ;准备返回值
    mov    esp, ebp                         ;撤销局部变量z
    pop    ebp                              ;撤销堆栈框架
    ret
```

3.2.2 逻辑运算指令

➢关于逻辑运算指令的通用说明

- ✓ 只有通用寄存器或存储单元可作为目的操作数，用于存放运算结果。
- ✓ 如只有一个操作数，则该操作数既是源又是目的。
- ✓ 如有两个操作数，那么最多只能有一个是存储单元，源操作数可以是立即数。
- ✓ 存储单元可采用各种存储器操作数寻址方式。
- ✓ 操作数可以是字节、字或者双字。如果有两个操作数，尺寸必须一致。

3.2.2 逻辑运算指令

➢否运算指令（NOT）

✓NOT指令的一般格式

```
NOT   OPRD
```

指令把操作数OPRD按位“取反”，然后送回OPRD。

✓使用举例

```
NOT   CL
NOT   EAX
NOT   BX
```

按位“取反”指，
把为0的位置成1，把为1的位清成0。

3.2.2 逻辑运算指令

➢与运算指令（AND）

✓AND指令的一般格式

```
AND  DEST，SRC
```

指令对两个操作数进行按位的逻辑“与”运算，结果送到目的操作数DEST。

按位“与”指，

当两个操作数对应位都为1，把结果的对应位设置成1，否则清成0。

✓使用举例

```
AND   ECX，ESI
MOV   AX，3437H          ;AX=3437H
AND   AX，0F0FH          ;AX=0407H
```

3.2.2 逻辑运算指令

➢或运算指令（OR）

✓OR指令的一般格式

```
OR   DEST，SRC
```

指令对两个操作数进行按位的逻辑“或”运算，结果送到目的操作数DEST。

按位“或”指，

当两个操作数对应位都为0，把结果的对应位清成0，否则设置成1。

✓使用举例

```
OR      CL，CH
OR      EBX，EAX
MOV     AL，41H          ;AL=01000001B，后缀B表示二进制
OR      AL，20H          ;AL=01100001B
```

3.2.2 逻辑运算指令

➢异或运算指令（XOR）

✓XOR指令的一般格式

XOR　DEST，SRC

指令对两个操作数进行按位的逻辑“异或”运算，结果送到目的操作数DEST。

按位“异或”指，

对应位不同，把结果的对应位设置成**1**，否则把结果的对应位清成**0**。

✓使用举例

```
MOV    AL，34H          ;AL=00110100B，符号B表示二进制
MOV    BL，0FH          ;BL=00001111B
XOR    AL，BL           ;AL=00111011B
XOR    ECX，ECX         ;ECX=0，CF=0
```

自己与自己异或结果为**0**

3.2.2 逻辑运算指令

➢测试指令（TEST）

✓TEST指令的一般格式

```
TEST   DEST，SRC
```

- 类似指令AND，把两个操作数进行按位“与”，但结果不送到目的操作数DEST，仅仅影响状态标志。
- 指令执行以后，标志ZF、PF和SF反映运算结果，标志CF和OF被清0。

✓使用举例

```
TEST   AL，BL
TEST   EDX，ECX
```

3.2.2 逻辑运算指令

➢测试指令（TEST）

✓使用举例

检查AL中的位6和位2是否有一位为1

```
TEST    AL，01000100B          ;符号B表示二进制
```

随后，判断标志位ZF

3.2.2 逻辑运算指令

➢演示函数cf312（另一个目标代码）

使速度最大化
编译：

```
push   ebp
mov    ebp, esp                          ;建立堆栈框架
mov    ecx, DWORD PTR [ebp+8]            ;取出参数x
xor    eax, eax                           ; z=0
test   cl, 3                             ;测试x的低8位（x & 3）
jne    SHORT  LN1cf312
push   esi                               ;临时保存ESI
mov    esi, DWORD PTR [ebp+12]           ;取出参数y
not    esi                                ; ~y
lea    edx, DWORD PTR [ecx-5]            ; x - 5
or     edx, esi                          ;( x - 5 ) | ~y
pop    esi                               ;恢复ESI
je    SHORT  LN2cf312
```

3.2.2 逻辑运算指令

➢演示函数cf312（另一个目标代码）

```
LN1cf312:
    xor    ecx, 255                ; x ^ 255
    mov    eax, ecx                ;准备返回值
LN2cf312:
    pop    ebp                     ;撤销堆栈框架
    ret
```

续前页

3.2.3 移位指令

➢移位指令

✓ 移动方式

- 一般移位指令
- 循环移位指令
- 双精度移位指令

✓ 移动方向

- 左移
- 右移

✓ 移动位数

- **1**位
- **m**位

3.2.3 移位指令

➢一般移位指令

✓一般移位指令的助记符

- 算术左移指令　SAL（Shift Arithmetic Left）
- 逻辑左移指令　SHL（SHift logic Left）
- 算术右移指令　SAR（Shift Arithmetic Right）
- 逻辑右移指令　SHR（SHift logic Right）

相同：
算术左移SAL
逻辑左移SHL

3.2.3 移位指令

➢一般移位指令

✓一般移位指令的格式

```
SAL   OPRD，count
SHL   OPRD，count
SAR   OPRD，count
SHR   OPRD，count
```

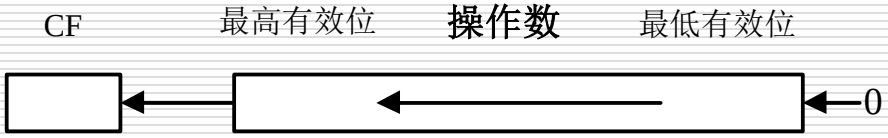
相同：
算术左移SAL
逻辑左移SHL

- 操作数OPRD可以是通用寄存器或存储器单元，尺寸可以是字节、字或者双字。
- count表示移位的位数，可以是一个8位立即数，可以是寄存器CL。寄存器CL表示移位数由CL的值决定。
- 通过截取count的低5位，实际的移位数被限于0到31之间。

3.2.3 移位指令

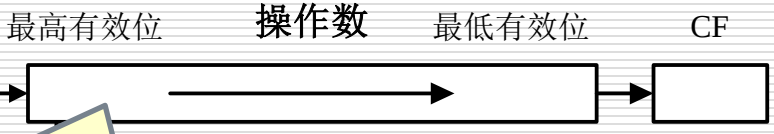
➢一般移位指令

✓一般移位指令的执行示意图



(a)逻辑左移/算术左移

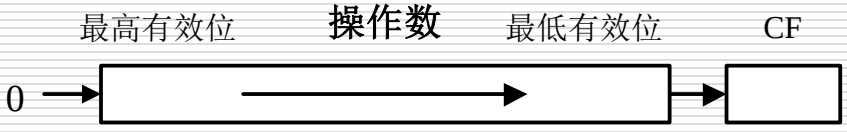
相同：
算术左移SAL
逻辑左移SHL



(b)算术右移

算术右移SAR

注意：没有在后面插入其他数



(c)逻辑右移

逻辑右移SHR

3.2.3 移位指令

➢一般移位指令

✓算术左移或逻辑左移指令SAL/SHL的示例

```
MOV    EBX, 7400EF9CH      ;EBX=7400EF9CH
ADD    EBX, 0              ;EBX=7400EF9CH，CF=0,SF=0,ZF=0,PF=1
SHL    EBX, 1              ;EBX=E801DF38H，CF=0,SF=1,ZF=0,PF=0
MOV    CL, 3               ;CL=3
SHL    EBX, CL             ;EBX=400EF9C0H，CF=1,SF=0,ZF=0,PF=1
SHL    EBX, 16             ;EBX=F9C00000H，CF=0,SF=1,ZF=0,PF=1
SHL    EBX, 12             ;EBX=00000000H，CF=0,SF=0,ZF=1,PF=1
```

观察被移位寄存器；观察状态标志

3.2.3 移位指令

➢一般移位指令

✓算术左移或逻辑左移指令SAL/SHL的示例

实现把寄存器AL中的内容（设为无符号数）乘以10

```
XOR    AH, AH          ;AH=0
SHL    AX, 1           ;2*X
MOV    BX, AX          ;暂存2*X
SHL    AX, 2           ;8*X
ADD    AX, BX          ;8*X+2*X
```

算术（逻辑）左移1位，相当于乘以2

3.2.3 移位指令

➢一般移位指令

✓算术右移指令SAR的示例

```
MOV    DX, 82C3H          ;DX=82C3H
SAR    DX, 1              ;DX=C161H,CF=1,SF=1,ZF=0,PF=0
MOV    CL, 3              ;CL=3
SAR    DX, CL             ;DX=F82CH,CF=0,SF=1,ZF=0,PF=0
SAR    DX, 4              ;DX=FF82H,CF=1,SF=1,ZF=0,PF=1
```

观察被移位寄存器；观察状态标志

算术右移**1**位，相当于有符号数除以**2**

3.2.3 移位指令

➢一般移位指令

✓逻辑右移指令SHR的示例

```
MOV    DX, 82C3H          ;DX=82C3H
SHR    DX, 1              ;DX=4161H,CF=1,SF=0,ZF=0,PF=0
MOV    CL, 3              ;CL=3
SHR    DX, CL             ;DX=082CH,CF=0,SF=0,ZF=0,PF=0
SHR    DX, 12             ;DX=0000H,CF=1,SF=0,ZF=1,PF=1
```

观察被移位寄存器；观察状态标志

逻辑右移**1**位，相当于无符号数除以**2**

3.2.3 移位指令

➢演示函数cf313

演示移位指令的使用

无符号整型

```
unsigned  cf313( unsigned x,  unsigned y )
{
    return  ( x << 2 ) - ( y >> 4 ) - ( x / 32)  - ( y * 8 );
}
```

3.2.3 移位指令

➢演示函数cf313

禁止优化 编译cf313 所得，目标代码

```
push    ebp
mov     ebp, esp                        ;建立堆栈框架
mov     eax, DWORD PTR [ebp+8]          ;取得参数x
shl     eax, 2                          ;把x向左移2位
mov     ecx, DWORD PTR [ebp+12]         ;取得参数y
shr     ecx, 4                          ;把y向右移4位
sub     eax, ecx
mov     edx, DWORD PTR [ebp+8]          ;取得参数y
shr     edx, 5                          ;无符号整数除以32
sub     eax, edx
mov     ecx, DWORD PTR [ebp+12]         ;取得参数y
shl     ecx, 3                          ;乘以8
sub     eax, ecx                        ;返回结在EAX中
pop     ebp                             ;撤销堆栈框架
ret
```

3.2.3 移位指令

➢循环移位指令

✓循环移位指令的助记符

- 左循环移位指令　**ROL（ROtate Left）**
- 右循环移位指令　**ROR（ROtate Right）**
- 带进位左循环移位指令　**RCL（Rotate Left through CF）**
- 带进位右循环移位指令　**RCR（Rotate Right through CF）**

3.2.3 移位指令

➢循环移位指令

✓循环移位指令的格式

```
ROL   OPRD，count
ROR   OPRD，count
RCL   OPRD，count
RCR   OPRD，count
```

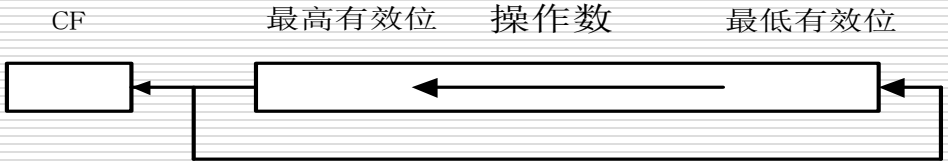
移位方式不同，
其他相同

- 操作数OPRD可以是通用寄存器或存储器单元，尺寸可以是字节、字或者双字。
- count表示移位的位数，可以是一个8位立即数，可以是寄存器CL。寄存器CL表示移位数由CL的值决定。
- 通过截取count的低5位，实际的移位数被限于0到31之间。

3.2.3 移位指令

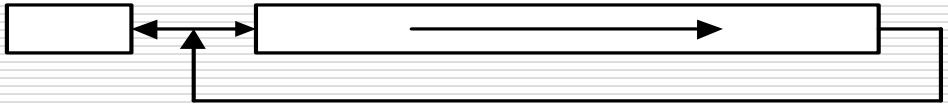
➢循环移位指令

✓循环移位指令的执行示意图



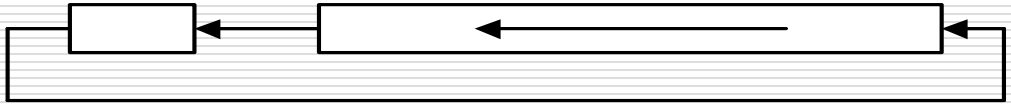
(a) 左循环移位指令ROL

左循环**ROL**



(b) 右循环移位指令ROR

右循环**ROR**



(c) 带进位左循环移位指 令RCL

带进位左循环**RCL**

(d) 带进位右循环移位指 令RCR

带进位右循环**RCR**

3.2.3 移位指令

➢循环移位指令

✓循环移位指令的示例

观察被移位寄存器；观察标志CF

```
MOV    DX, 82C3H          ;DX=82C3H
ROL    DX, 1              ;DX=0587H, CF=1
MOV    CL, 3              ;CL=3
ROL    DX, CL             ;DX=2C38H, CF=0
MOV    EBX, 8A2035F7H     ;EBX=8A2035F7H
ROR    EBX, 4             ;EBX=78A2035FH, CF=0
STC                       ;CF=1（设置进位标志）
RCL    EBX, 1             ;EBX=F14406BFH, CF=0
RCR    EBX, CL            ;EBX=DE2880D7H, CF=1
```

3.2.3 移位指令

➢循环移位指令

✓循环移位指令的示例

实现把**AL**的最低位送入**BL**的最低位，仍保持**AL**不变

```
ROR    BL, 1          ;BL循环右移1位
ROR    AL, 1          ;AL循环右移1位，最低位到CF
RCL    BL, 1          ;BL带进位左移，带进了来自AL的最低位
ROL    AL, 1          ;恢复AL
```

3.2.3 移位指令

把一个操作数的部分内容
移位方式复制到另一个操作数

➢双精度移位指令

✓双精度移位指令的格式

```
SHLD  OPRD1，OPRD2，count
SHRD  OPRD1，OPRD2，count
```

- 操作数OPRD1作为目的操作数，可以是通用寄存器或者存储器单元，尺寸是字或者双字。
- 操作数OPRD2相当于源操作数，只能寄存器，尺寸必须与操作数OPRD1一致。OPRD2本身不会发生变化。
- count表示移位的位数，可以是一个8位立即数，也可以是寄存器CL。寄存器CL表示移位数由CL的值决定。通过截取count的低5位，移位数被限于0到31之间。

3.2.3 移位指令

➢双精度移位指令

✓双精度移位指令的示例

```
MOV    AX, 8321H              ;AX=1000001100100001B
MOV    DX, 5678H              ;DX=0101011001111000B
SHLD   AX, DX, 1              ;AX=0642H, DX=5678H, CF=1, OF=1
SHLD   AX, DX, 2              ;AX=1909H, DX=5678H, CF=0, OF=0
;
MOV    EAX, 01234867H
MOV    EDX, 5ABCDEF9H
SHRD   EAX, EDX, 4            ;EAX=90123486H, CF=0, OF=1
MOV    CL, 8
SHRD   EAX, EDX, CL           ;EAX=F9901234H, CF=1, OF=0
```

3.2.3 移位指令

➢双精度移位指令

✓双精度移位指令的示例

实现把**EAX**中的**32**位数，保存到寄存器对**DX:AX**

```
SHLD    EDX，EAX，16
```